Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Evaluating and Implementing Hybrid Approach for SPECFEM3D_GLOBE

Marcin Zielinski[1a], John Donners[a]

[a]*SARA B.V., Science Park 140, 1098XG Amsterdam, The Netherlands*

Abstract

The entire project focused on an evaluation of the code for a possible introduction of OpenMP and its actual implementation and extensive tests. Major time consuming parts of the code were detected and thoroughly analyzed. The most time consuming part was successfully parallelized using OpenMP. Very extensive test simulations using the hybrid code allowed for many further improvements and validations of its results. Possible improvements have also been discussed with the developers to be implemented in the near future.

1. Introduction

The SPECFEM3D_GLOBE application [1], which simulates three-dimensional global and regional (continental-scale) seismic wave propagation based on the spectral-element method (SEM) [2], is a very popular scientific tool within its community. The code has been parallelized very efficiently with MPI winning the Gordon Bell award for best performance at the SuperComputing 2003 in Phoenix, Arizona (USA) [3]. It was a finalist again in 2008 for a run at 0.16 petaflops (sustained) on 149,784 processors of the 'Jaguar' Cray XT5 system at Oak Ridge National Laboratories (USA) [4]. It also won the BULL Joseph Fourier supercomputing award in 2010. However, due to its nature the application uses entirely static arrays with fixed dimensions that renders the whole application a bit inflexible in use. The application has been divided in two parts called '*mesher*' (meshfem3D) which creates the three-dimensional mesh of the Earth and '*solver*' (specfem3D) which calculates synthetic seismograms in that three-dimensional earth model. The hybrid approach (MPI with OpenMP) has been introduced to the *solver* application.

[1] Corresponding author. *E-mail address*: Marcin.Zielinski@sara.nl

The split of the application between the *mesher* and the *solver* requires rerunning of the preprocessor (*mesher*) when the number of MPI tasks changes, which, moreover, entails recompilation of the entire application. The application divides the three-dimensional mesh using five main parameters which are set explicitly by the user in the input file. The restrictions put on these parameters cause the inflexibility in choosing number of the MPI tasks, which can be also changed by the user him self. The hybrid version of the *solver* application should in principle add more freedom to how many parallel tasks can be run and give a better scaling when moved towards petaflop supercomputers to run simulations of high accuracy and bigger scale.

1.1. Configuration

For the hybrid part of the *SPECFEM3D_GLOBE* application, *solver*, a major part of the execution time is spent computing internal forces in the solid regions. There is however no real set of input data but parameters describing the receiving stations. Each simulation runs over a period of time divided into time steps.

The amount of degrees of freedom being computed at each time step is dependent on the five main parameters set in the input file. The rough estimates of the amount of degrees of freedom, being calculated at each time step, vary between $3 \cdot 10^{08}$ to $5 \cdot 10^{08}$ of single precision type, for simulations performed during the validation, performance and scaling tests. The time taken to run *solver* is also strictly dependent on the number of the total degrees of freedom, on the number of the MPI tasks, and on the number of the time steps of the entire simulation. For the test simulations considered in this paper, the total *solver* execution time may vary between 15 to even 22 hours, assuming simulation runs with the smallest number of degrees of freedom and with the lowest number of CPU cores used.

2. Performance analysis

The profiling analysis, performed before the hybridization work, of the original MPI code has shown that the most intensive and time consuming part of the *solver* is a subroutine called *compute_forces_crust_mantle_Dev*. The time spent on the execution of this subroutine varies between 85 and up to even 90% of the total *solver* execution time. The subroutine calculates and updates the acceleration vector, *accel_crust_mantle*, which is a second order derivative of displacement vector (*displ_crust_mantle*) with respect to time, used to calculate the mass matrix and solve the equation of motion in crust and mantle[5,6]. The calculations are perform in a loop over the spectral elements within the given mesh slice. Each of the mesh slices share boundary grid points with a neighboring mesh slice for which data needs to be exchanged using MPI communication. The MPI communication accommodates for around 0.005 to 0.01% of the total execution time of the *compute_forces_crust_mantle_Dev* subroutine and for around 0.1 to 0.25% of the entire MPI communication within the *solver*. Any possible gains from using significantly less MPI tasks, i.e. one master MPI task per node, will likely introduce no practical gain in the execution time of the *solver*.

To analyze the performance and the scaling of the hybrid, *solver* part, special test simulations were performed. The tests were divided into two separate simulation runs within the same model, but with different parameters describing it. Simulations #1 were run on high-resolution, regional scale with jobs having various number of MPI tasks (36, 64, 144, 256, 576, 2304) and with varying number of OpenMP threads per each given job. Simulations #2 where run, on the other hand, on global scale and also with jobs having various number of MPI tasks (150, 600, 2400) and also with varying number of OpenMP threads per each given job. The system used for testing was an IBM Power6 based machine.

The *compute_forces_crust_mantle_Dev* subroutine is called four times per each time step. For that reason, it has been also considered to implement orphaning, thus creating a parallel region before the first call and ending it after the last call. A working version with such solution exhibited undesirable effects. A possible gain from allocating the parallel region only once per four calls has been completely canceled out by effects due to bigger amount of MPI

communication in between the calls and many explicit and implicit barriers set to synchronize the OpenMP threads. In summary, the orphaned version of the hybrid code turned out to run incredibly slower than the regular hybrid code and than the pure MPI version.

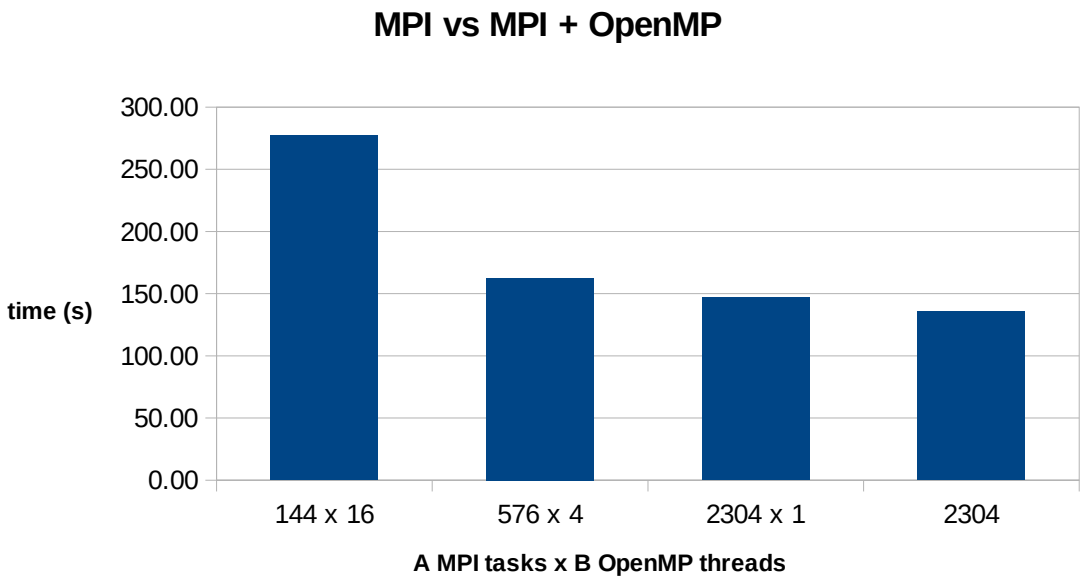

Fig. 1. Performance scaling plot for a varying number of MPI tasks and threads, with equal total number of used cores. Simulations are performed on the Power6 testing machine for Simulations #1.

The Simulations #1 set of performance and scaling tests consisted of 29 calculations, the simulations #2 set of 13 calculations. For each specified number of MPI tasks, a set of four or five subjobs were run with varying number of OpenMP threads. Figure 1 indicates how the hybrid combination of varying number MPI tasks and varying number of OpenMP threads, giving in total the same amount cores used, performs on the test machine. Figure 2 presents similar trend for the Simulations #2 set. In both test sets there is a noticeable good behavior and timing in the case of four OpenMP threads with comparison to the original MPI code. This is most likely due to the architecture of the test machine which has 4 Multi-Chip Modules (MCM) with 4 CPUs on each MCM. Both of these calculations (576 x 4 and 600 x 4) are approximately 20 to 25% slower than the original MPI version for the same total number of parallel 'tasks' (2304 and 2400 respectively). Possibly a different behavior might be observed when applied to a different architecture. It is however worth exploring a possible scaling of this case, for both #1 and #2 simulations. Figure 5 and 6 presents scaling results of simulations #1 test with 576 MPI tasks and simulations #2 test with 600 MPI tasks, respectively.

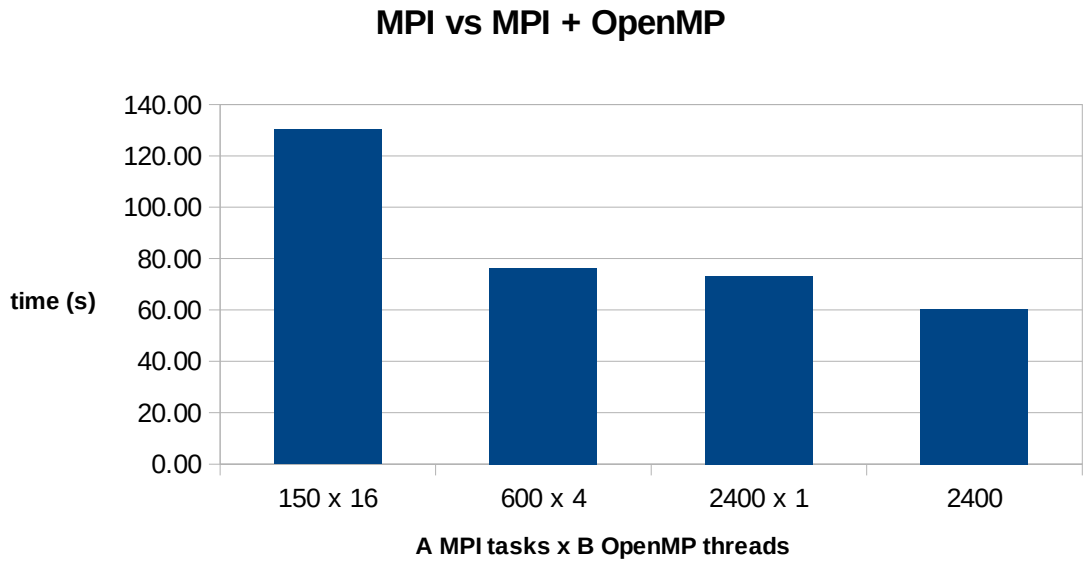

Fig. 2. Performance scaling plot for a varying number of MPI tasks and threads, with equal total number of used cores. Simulations are performed on the Power6 testing machine for Simulations #2.

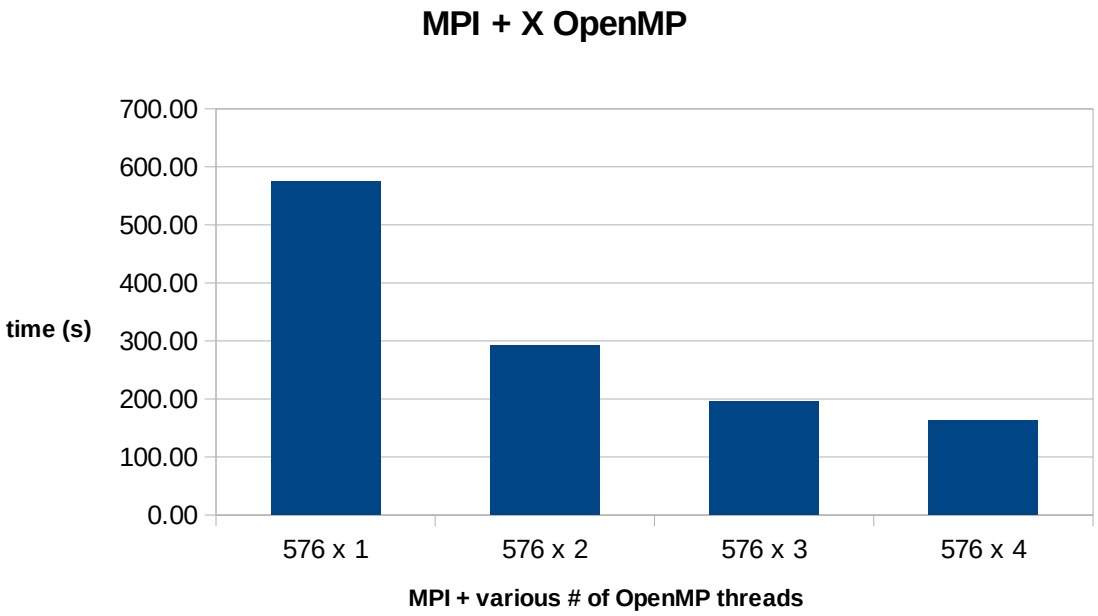

Fig. 3. Performance scaling of Simulations #1 test with 576 MPI tasks and increasing number of OpenMP threads on the Power6 testing machine.

In principle the hybrid code, i.e. with 2300 MPI tasks and 1 OpenMP thread, should run as fast as the pure MPI example with 2300 MPI tasks. The small timing differences between the two are yet unknown and the issue is still being investigated.

Both Figures, 3 and 4, are exhibiting a positive scaling trend. It needs to be reminded that the OpenMP parallel region concerns, at most, 90% of the total *solver* execution time. Therefore a possible speed-up between one and two thread could be of, at most, 45%, or 30% between one and three threads or 22.5% between one and four threads. For both scaling tests, these ratios are almost in line showing a good scaling behavior of the hybrid code.

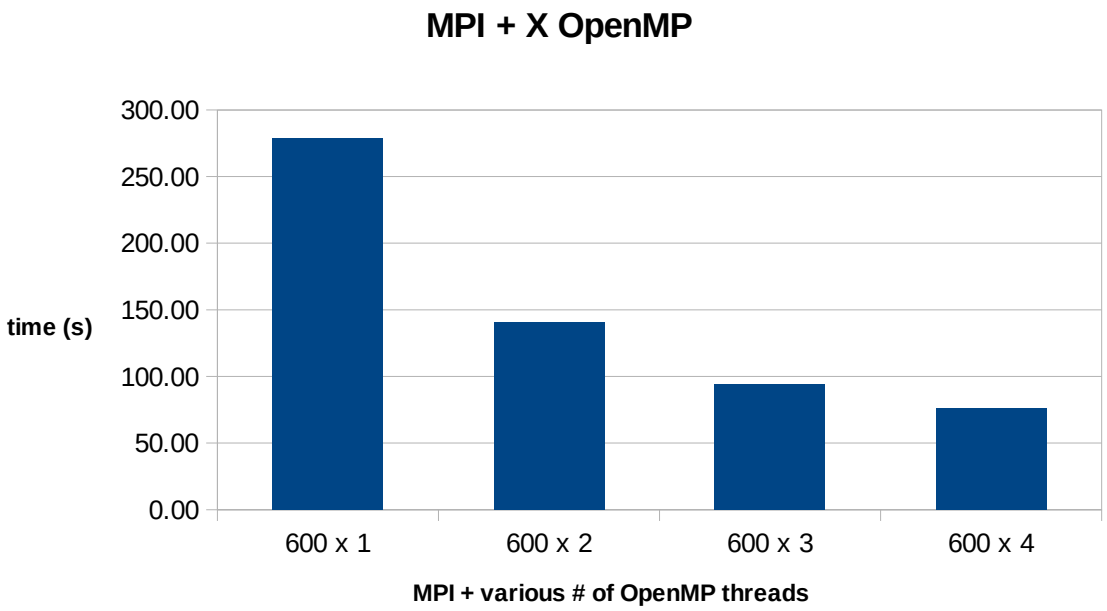

Fig. 4. Performance scaling of Simulations #2 test with 600 MPI tasks and increasing number of OpenMP threads on the Power6 testing machine.

3. Implementing OpenMP

The OpenMP threads support implementation process, into the *solver* part of the SPECFEM3D_GLOBE application, has started with a profiling analysis. The entire profiling analysis has been done using SCALASCA. The first global *solver* analysis has shown that one particular subroutine, called *compute_forces_crust_mantle_Dev*, accounted for 85 till 90% of the entire *solver* execution time. A more detailed (manual instrumentation) analysis of the *compute_forces_crust_mantle_Dev* subroutine exhibited one, smallest and indivisible part of that subroutine, a DO-loop, which accommodated for 30 to 35% of the entire *solver* execution time. The first OpenMP elements were introduced to that particular DO-loop. Although the DO-loop accommodates for only 35% of the entire *solver* execution time, it covers major part of the subroutine and most of the subroutines variables had to be defined within the parallel OpenMP region. Each variable, that is passed in to the subroutine as an argument has been defined as SHARED. One of these variables has its values set inside the parallel region and upon meeting a specific IF condition, available OpenMP threads may write to the same cell of this variable, on the same time. Therefore, the writing occurs within a CRITICAL region to avoid racing conditions. Every variable declared only inside the *compute_forces_crust_mantle_Dev* subroutine has been defined within the parallel region as PRIVATE, since they are used as an intermediate storing arrays to compute the final accelerator vector, outside the parallel region.

The validation and testing of the partially hybridized *compute_forces_crust_mantle_Dev* subroutine, using full-length scale simulation runs, has shown no deviations in computed seismograms with respect to the original, pure

MPI code. The code did not exhibit any undesirable behaviors hence the next step in hybridization was to cover the entire subroutine with a one big parallel OpenMP region. The declaration of SHARED and PRIVATE variables has been done based on similar rules as before. Out of all the variables declared within the parallel region as SHARED, only four of them are being modified inside. One of them, mentioned before with respect to the particular DO-loop, was put inside a CRITICAL region to avoid possible racing conditions. The three remaining SHARED variables, which are modified inside the parallel region, depend strictly on the outermost DO-loops index (*ispec*) which takes independent and different values for each OpenMP thread hence operating on separate sets of data. Therefore it is not necessary to contain these variables within the CRITICAL regions.

The parallelization process was very time consuming. The entire OpenMP region inside the *compute_forces_crust_mantle_Dev* subroutine covers over one hundred variables. Each variable had to be checked and properly allocated as SHARED or PRIVATE. The possibility of making an easy mistake with the allocation process was very high, therefore the process had to be very thorough. Validation runs had to be performed to confirm those allocations. Moreover, numerous test runs with different scheduling policies, chunk sizes and number of OpenMP threads were run to find the optimal settings.

Confiding the entire subroutine, which starts immediately with an outermost DO-loop, in one parallel region should in principle have its advantages. In this particular case however, the parallel region contains calls to a MPI-communication subroutines, which takes place every 1500th iteration of the DO-loop straight at its beginning. The MPI communication occurs only on the MASTER thread and an OpenMP BARRIER, just before and after each MPI call, assures a full synchronization between the threads. The profiling analyses done on the hybrid code have shown, that explicit placement of the BARRIER has no real impact on the performance of the hybrid code and accounts insignificantly for less than 0.001% of total *solver* execution time.

Every proper implementation of OpenMP into already existing MPI code, with MPI communication inside the parallel OpenMP region, requires the MPI code to become thread-aware. A proper thread level of support needs to be selected and determined upon the start of the parallel MPI environment and this can be only achieved by using MPI_INIT_THREAD initialization call in place of the regular MPI_INIT call. The MPI_INIT_THREAD takes two variables as calling parameters, one selecting the thread level of support and second, determining the highest possible thread level of support, for the given available MPI implementation. The MPI calling subroutines, inside *compute_forces_crust_mantle_Dev* subroutine, are completely independent on the division of workload and data between the threads, meaning, the communication occurs for MPIs entire set of data. Based on that, the thread level of support was selected to be MPI_THREAD_FUNNELED. It means, that the communication is funneled to the master threads, hence occurs only on the master threads while the other threads are idle. Moreover, as previously mentioned, the MPI calling subroutines have been confined inside OpenMP MASTER regions and an explicit OpenMP BARRIERs put before and after them to assure the data consistency.

The inner DO-loop, parallelized with the OpenMP threads, has been given a SCHEDULE property with a 'runtime' value, to dispatch the work load between the threads. The 'runtime' value allows to choose the scheduling policy from the run time environment upon execution of the application, thus making it more flexible for the end-user when used on different, than the one used during development, system architectures. It has been noted, based on broader test jobs and analyses, that the preferred scheduling policy, for IBM Power6 based system running SLES11 SP1 operating system, is DYNAMIC with size of the chunk being at least of the same size as the number of OpenMP threads, but preferably being twice as big as that number.

The correctness of the code can be measured and checked only by comparing two seismograms, i.e. from the hybrid and the original MPI code. The seismograms have to be obtained within the same model simulation, defined with the same main parameters and only for simulations run over an extensive period of time, i.e. with RECORD_LENGTH_IN_MINUTES value being greater than or equal to 15,0. These strict rules have been followed when the correctness of the hybrid code has been assessed. No differences in seismograms have been noted between the hybrid and the original MPI code, hence the resulting hybrid code has been assumed safe and valid.

The mentioned timing differences between one-threaded hybrid code and the pure MPI code may be however minimized with further OpenMP parallelization of the other *solver* parts. The possible parallelization of the other DO-loops will be significantly less difficult to implement than that of *compute_forces_crust_mantle_Dev*.

4. Conclusions

The final hybrid code of SPECFEM3D_GLOBE has proven to scale flawlessly with almost linear accuracy. The minimal timing difference between the pure MPI code and the hybrid code with only one thread is as of yet unknown and still under investigation. The resulting seismographs computed by the hybrid code are correct. During the parallelization process, a very tight and flawless communication with the main developers was maintained. The resulting hybrid OpenMP code has been already merged with the official SVN trunk version of SPECFEM3D_GLOBE. Possible new improvements to the hybrid code were also discussed with the main developers.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-211528 and FP7-261557. The work is achieved using the PRACE Research Infrastructure resources CURIE at TGCC, Bruyères-Le-Châtel, France and Huygens at SARA, Amsterdam, The Netherlands.

References

1. SPECFEM3D_GLOBE official web site `http://www.geodynamics.org/cig/software/specfem3d-globe`
2. A. T. Patera, A spectral element method for fluid dynamics - Laminar flow in a channel expansion. Journal of Computational Physics, 54:468—488, 1984
3. D. Komatitsch, S. Tsuboi, C. Ji, and J. Tromp. A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth Simulator. Proceedings of the ACM/IEEE Supercomputing SC'2003 conference, pages 4–11, 2003. doi: 10.1109/SC.2003.10023. Gordon Bell Prize winner article.
4. L. Carrington, D. Komatitsch, M. Laurenzano, M. Tikir, D. Michéa, N. Le Goff, A. Snavely, and J. Tromp. High frequency simulations of global seismic wave propagation using SPECFEM3D_GLOBE on 62 thousand processor cores. Proceedings of the ACM/IEEE Supercomputing SC'2008 conference, pages 1–11, 2008. doi: 10.1145/1413370.1413432. Article #60, Gordon Bell Prize finalist article.
5. D. Komatitsch and J. Tromp, Introduction to the spectral element method for three-dimensional seismic wave propagation. Geophysical Journal International, 139:806-822, 1999
6. D. Komatitsch, S. Tsuboi and J. Tromp, The Spectra-Element Method in Seismology. Geophysical Monograph, 157:255, 2005